Multithreaded Implicitly Dealiased Pseudospectral Convolutions

Malcolm Roberts and John C. Bowman

University of Alberta

May 2012

Pseudospectral simulations

- The incompressible 2D vorticity formulation

$$\frac{\partial \omega}{\partial t} + (\boldsymbol{u} \cdot \nabla)\,\omega = \nu \nabla^2 \omega$$

Pseudospectral simulations

- The incompressible 2D vorticity formulation

$$\frac{\partial \omega}{\partial t} + (\boldsymbol{u} \cdot \nabla)\, \omega = \nu \nabla^2 \omega$$

is Fourier-transformed into

Pseudospectral simulations

- The incompressible 2D vorticity formulation

$$\frac{\partial \omega}{\partial t} + (u \cdot \nabla)\,\omega = \nu \nabla^2 \omega$$

is Fourier-transformed into

$$\frac{\partial \omega_k}{\partial t} = \sum_{p+q=k} \frac{\epsilon_{kpq}}{q^2} \omega_p^* \omega_q^* - \nu k^2 \omega_k$$

$$\epsilon_{kpq} = (\hat{z} \cdot p \times q)\delta(k + p + q)$$

Pseudospectral simulations

- The incompressible 2D vorticity formulation

$$\frac{\partial \omega}{\partial t} + (u \cdot \nabla)\,\omega = \nu \nabla^2 \omega$$

is Fourier-transformed into

$$\frac{\partial \omega_k}{\partial t} = \sum_{p+q=k} \frac{\epsilon_{kpq}}{q^2} \omega_p^* \omega_q^* - \nu k^2 \omega_k$$

$$\epsilon_{kpq} = (\hat{z} \cdot p \times q)\delta(k+p+q)$$

- The nonlinearity becomes a convolution:

$$(F * G)_k = \sum_{k_1,k_2} F_{k_1} G_{k_2}\, \delta_{k,k_1,k_2}.$$

Non-centered data

- Input data: $\{F_k\}_{k=0}^{N-1}$ and $\{G_k\}_{k=0}^{N-1}$.

Non-centered data

- Input data: $\{F_k\}_{k=0}^{N-1}$ and $\{G_k\}_{k=0}^{N-1}$.
- This produces non-centered convolutions:

$$(F * G)_k = \sum_{\ell=0}^{k} F_\ell G_{k-\ell}$$

Non-centered data

- Input data: $\{F_k\}_{k=0}^{N-1}$ and $\{G_k\}_{k=0}^{N-1}$.
- This produces non-centered convolutions:

$$(F * G)_k = \sum_{\ell=0}^{k} F_\ell G_{k-\ell}$$

- For non-centered data, $*(F, G, H) = F * (G * H)$.

Centered data

- Input data: $\{F_k\}_{k=-N+1}^{N-1}$ and $\{G_k\}_{k=-N+1}^{N-1}$.

Centered data

- Input data: $\{F_k\}_{k=-N+1}^{N-1}$ and $\{G_k\}_{k=-N+1}^{N-1}$.

$$(F*G)_k = \sum_{\ell=\max(-N+1,k-N+1)}^{\min(N-1,k+N-1)} F_\ell G_{k-\ell}$$

Centered data

- Input data: $\{F_k\}_{k=-N+1}^{N-1}$ and $\{G_k\}_{k=-N+1}^{N-1}$.

$$(F * G)_k = \sum_{\ell=\max(-N+1,k-N+1)}^{\min(N-1,k+N-1)} F_\ell G_{k-\ell}$$

- Considering Hermitian-symmetric data ($F_{-k} = F_k^*$), we compute data for $k \geq 0$, so

$$(F * G)_k = \sum_{\ell=k-N+1}^{N-1} F_\ell G_{k-\ell}.$$

Centered data

- Input data: $\{F_k\}_{k=-N+1}^{N-1}$ and $\{G_k\}_{k=-N+1}^{N-1}$.

$$(F * G)_k = \sum_{\ell=\max(-N+1,k-N+1)}^{\min(N-1,k+N-1)} F_\ell G_{k-\ell}$$

- Considering Hermitian-symmetric data ($F_{-k} = F_k^*$), we compute data for $k \geq 0$, so

$$(F * G)_k = \sum_{\ell=k-N+1}^{N-1} F_\ell G_{k-\ell}.$$

- For centered data, $*(F, G, H) \neq F * (G * H)$.

FFT-based convolutions

- The convolution sum involves $\mathcal{O}(N^2)$ terms. Using FFTs, we can compute a convolution in $\mathcal{O}(N \log N)$ operations.

FFT-based convolutions

- The convolution sum involves $\mathcal{O}(N^2)$ terms. Using FFTs, we can compute a convolution in $\mathcal{O}(N \log N)$ operations.
- FFTs produce cyclic convolutions. Linear convolutions are attained if one zero-pads the input data.

FFT-based convolutions

- The convolution sum involves $\mathcal{O}(N^2)$ terms. Using FFTs, we can compute a convolution in $\mathcal{O}(N \log N)$ operations.
- FFTs produce cyclic convolutions. Linear convolutions are attained if one zero-pads the input data.
- Non-centered data is padded from length N to length $2N$.

FFT-based convolutions

- The convolution sum involves $\mathcal{O}(N^2)$ terms. Using FFTs, we can compute a convolution in $\mathcal{O}(N \log N)$ operations.
- FFTs produce cyclic convolutions. Linear convolutions are attained if one zero-pads the input data.
- Non-centered data is padded from length N to length $2N$.
- Centered data is padded from length $2N-1$ to length $3N$.

Implicit Zero-padding

Implicit padding involves using a separate work array to compute the DFT:

$$f_x = \sum_{k=0}^{2N-1} \zeta_{2N}^{xk} F_k, \quad F_k = 0 \text{ if } k \geq N$$

Implicit Zero-padding

Implicit padding involves using a separate work array to compute the DFT:

$$f_x = \sum_{k=0}^{2N-1} \zeta_{2N}^{xk} F_k, \quad F_k = 0 \text{ if } k \geq N$$

is attained by computing

Implicit Zero-padding

Implicit padding involves using a separate work array to compute the DFT:

$$f_x = \sum_{k=0}^{2N-1} \zeta_{2N}^{xk} F_k, \quad F_k = 0 \text{ if } k \geq N$$

is attained by computing

$$f_{2x} = \sum_{k=0}^{N-1} \zeta_N^{xk} F_k$$

Implicit Zero-padding

Implicit padding involves using a separate work array to compute the DFT:

$$f_x = \sum_{k=0}^{2N-1} \zeta_{2N}^{xk} F_k, \quad F_k = 0 \text{ if } k \geq N$$

is attained by computing

$$f_{2x} = \sum_{k=0}^{N-1} \zeta_N^{xk} F_k$$

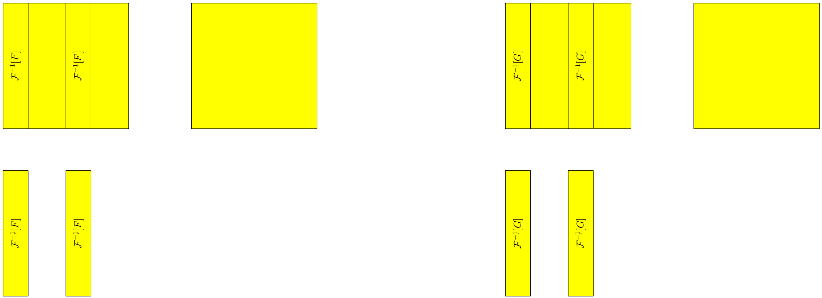
and

$$f_{2x+1} = \sum_{k=0}^{N-1} \zeta_N^{xk} (\zeta_{2N}^{x} F_k)$$

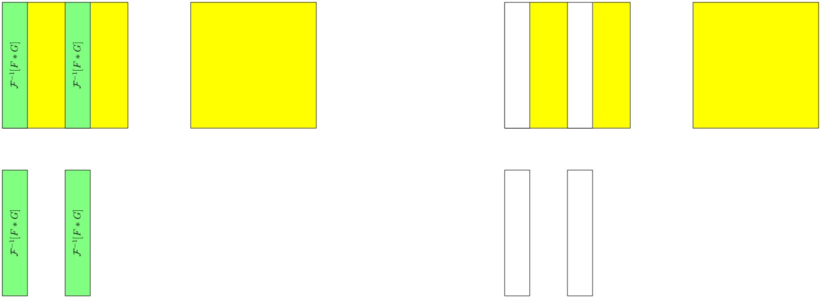
Implicit Zero-padding

Implicit Zero-padding

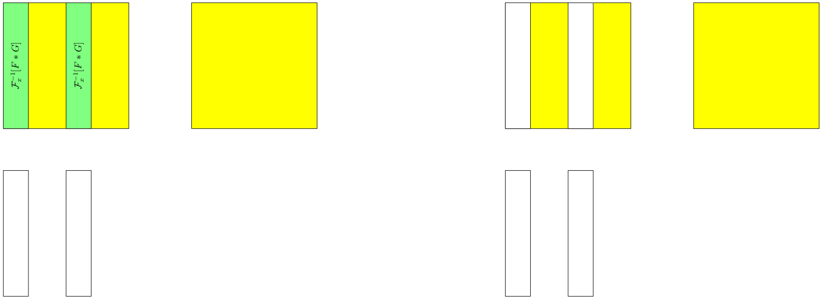
Implicit Zero-padding



Implicit Zero-padding



Implicit Zero-padding



Implicit Zero-padding

Implicit Zero-padding

Memory requirements

Work memory required for an n-dimensional non-centered convolution:

n	Explicit	Implicit
1	$2N_x$	$2N_x$
2	$6N_xN_y$	$2N_xN_y + 2PN_y$
3	$14N_xN_yN_z$	$2N_xN_yN_z + 2PN_yN_z$

Memory requirements

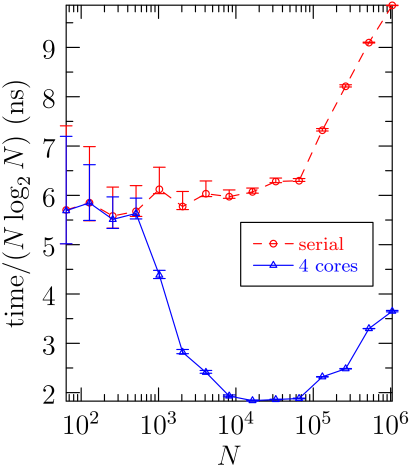
Work memory required for an n-dimensional non-centered convolution:

n	Explicit	Implicit
1	$2N_x$	$2N_x$
2	$6N_xN_y$	$2N_xN_y + 2PN_y$
3	$14N_xN_yN_z$	$2N_xN_yN_z + 2PN_yN_z$

Work memory required for an n-dimensional centered convolution:

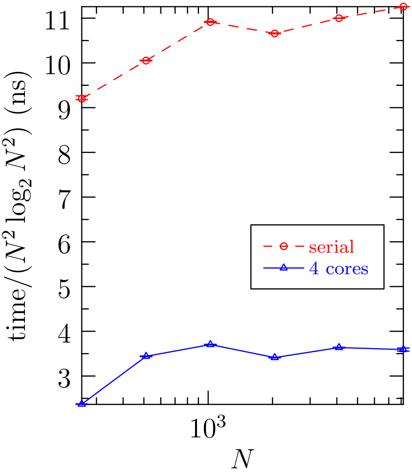
n	Explicit	Implicit
1	$2N_x$	$2N_x$
2	$5N_xN_y$	$2N_xN_y + PN_y$
3	$19N_xN_yN_z$	$4N_xN_yN_z + 2PN_xN_y$

Performance: multiple threads



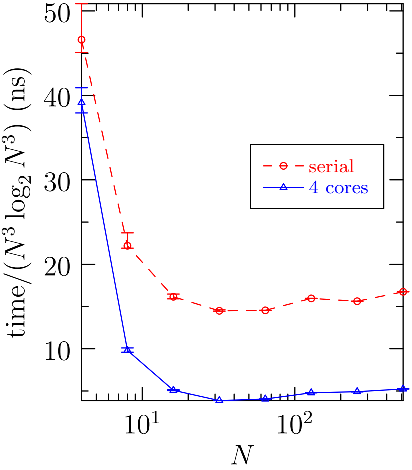
Non-centered 1D convolution.

Performance: multiple threads



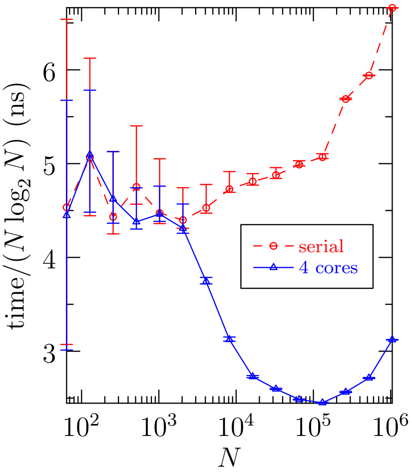
Non-centered 2D convolution.

Performance: multiple threads



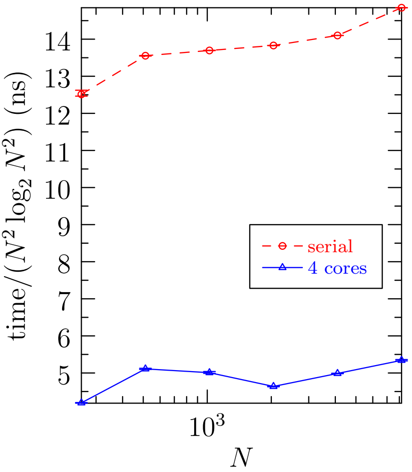
Non-centered 3D convolution.

Performance: multiple threads



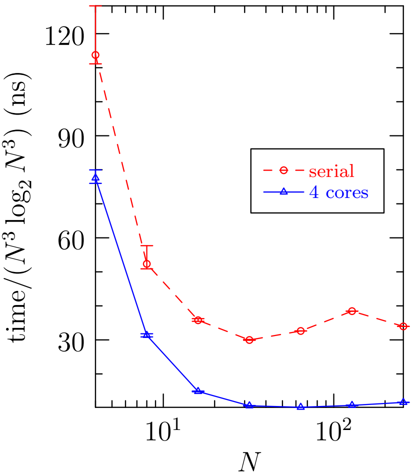
Centered 1D convolution.

Performance: multiple threads



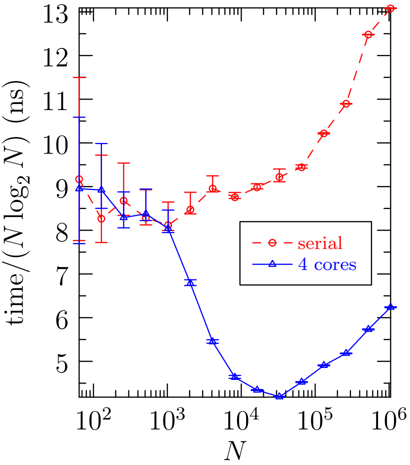
Centered 2D convolution.

Performance: multiple threads



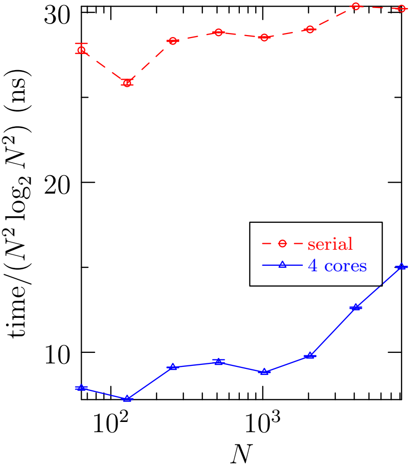
Centered 3D convolution.

Performance: multiple threads



Centered ternary 1D convolution.

Performance: multiple threads



Centered ternary 2D convolution.

Performance: multiple threads

- One-dimensional convolutions on four cores are about 2 times as fast as on one core.

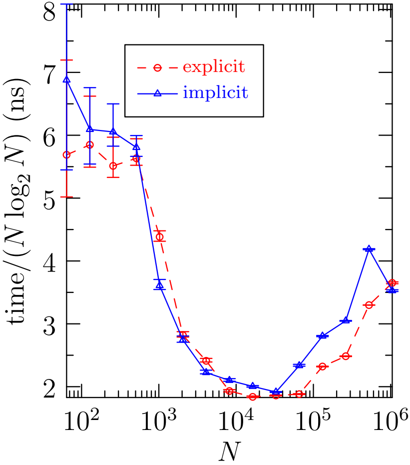
Performance: multiple threads

- One-dimensional convolutions on four cores are about 2 times as fast as on one core.
- Two-dimensional convolutions on four cores are about 3 times as fast.

Performance: multiple threads

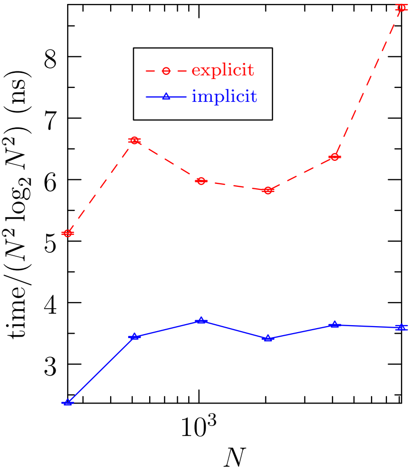
- One-dimensional convolutions on four cores are about 2 times as fast as on one core.
- Two-dimensional convolutions on four cores are about 3 times as fast.
- Three-dimensional convolutions on four cores are about 3.5 times as fast.

Performance: explicit vs. implicit



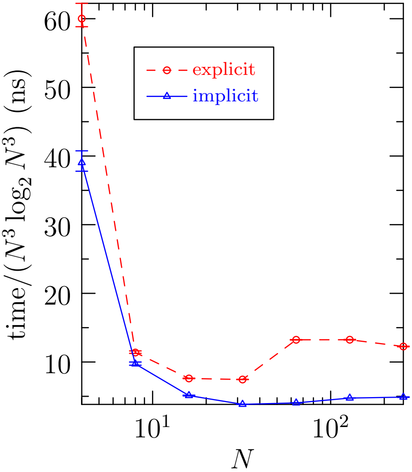
Non-centered 1D convolution.

Performance: explicit vs. implicit



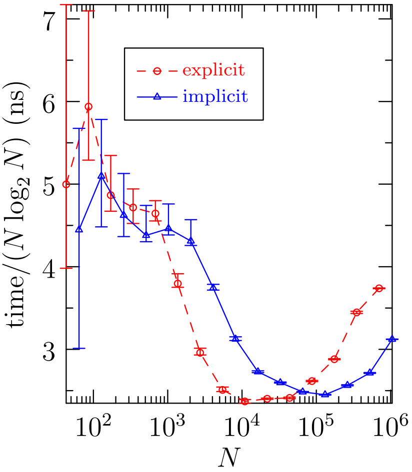
Non-centered 2D convolution.

Performance: explicit vs. implicit



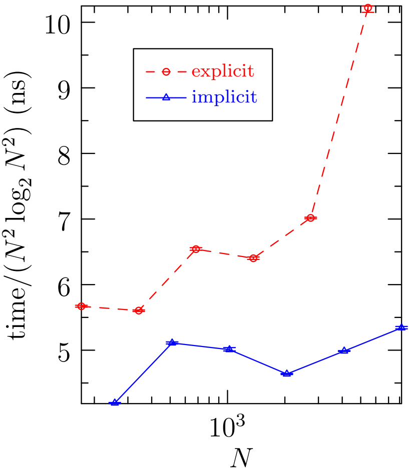
Non-centered 3D convolution.

Performance: explicit vs. implicit



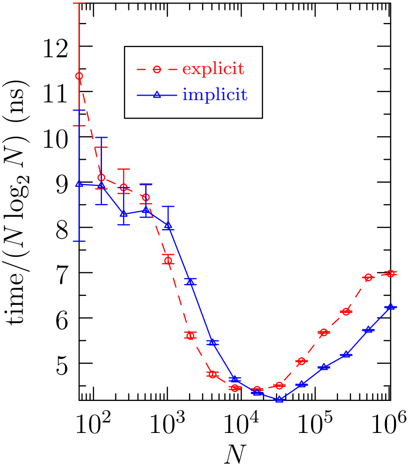
Centered 1D convolution.

Performance: explicit vs. implicit



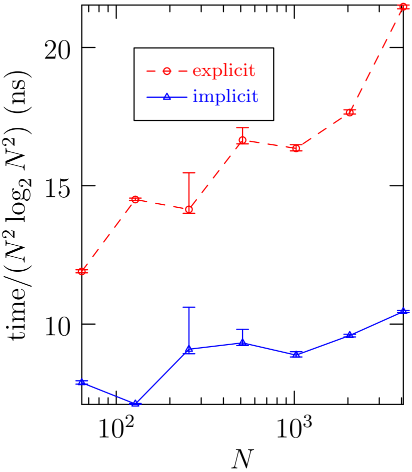
Centered 2D convolution.

Performance: explicit vs. implicit



Centered ternary 1D convolution.

Performance: explicit vs. implicit



Centered ternary 2D convolution.

Summary of Results

- Implicit methods require much less work memory than is required by explicit methods .

Summary of Results

- Implicit methods require much less work memory than is required by explicit methods .
- The implicit method had a speedup of up to 3.6 on four cores, while the explicit method sped-up of up to a factor of 3.

Summary of Results

- Implicit methods require much less work memory than is required by explicit methods .
- The implicit method had a speedup of up to 3.6 on four cores, while the explicit method sped-up of up to a factor of 3.
- The implicit method is around twice as fast as the explicit method for multidimensional convolutions.

Usage example

Computing the nonlinear source of the 2D incompressible Navier–Stokes equations in a vorticity formulation, which appears in Fourier space as

$$\sum_{\mathbf{p}} \frac{p_x k_y - p_y k_x}{|\mathbf{k}-\mathbf{p}|^2}\omega_{\mathbf{p}}\omega_{\mathbf{k}-\mathbf{p}},$$

Usage example

Computing the nonlinear source of the 2D incompressible Navier–Stokes equations in a vorticity formulation, which appears in Fourier space as

$$\sum_{\mathbf{p}} \frac{p_x k_y - p_y k_x}{|\mathbf{k}-\mathbf{p}|^2} \omega_{\mathbf{p}} \omega_{\mathbf{k}-\mathbf{p}},$$

is performed as follows:

$$\texttt{conv2}(ik_x\omega, ik_y\omega, ik_y\omega/k^2, -ik_x\omega/k^2).$$

Usage example

Computing the nonlinear source of the 2D incompressible Navier–Stokes equations in a vorticity formulation, which appears in Fourier space as

$$\sum_{\mathbf{p}} \frac{p_x k_y - p_y k_x}{|\mathbf{k} - \mathbf{p}|^2} \omega_{\mathbf{p}} \omega_{\mathbf{k}-\mathbf{p}},$$

is performed as follows:

$$\texttt{conv2}(ik_x\omega, ik_y\omega, ik_y\omega/k^2, -ik_x\omega/k^2).$$

One also has the option of passing work arrays to `conv2`, `which can then be used elsewhere.`

Conclusion

- Implicitly zero-padding multi-dimensional convolutions is faster and requires less memory than explicit routines.

Conclusion

- Implicitly zero-padding multi-dimensional convolutions is faster and requires less memory than explicit routines.
- The algorithm has been successfully implemented on a shared-memory architecture with only a small increase in work memory.

Conclusion

- Implicitly zero-padding multi-dimensional convolutions is faster and requires less memory than explicit routines.
- The algorithm has been successfully implemented on a shared-memory architecture with only a small increase in work memory.
- Convolution algorithms are available for complex non-centered data and centered Hermitian-symmetric data in 1D, 2D, and 3D.

Conclusion

- Implicitly zero-padding multi-dimensional convolutions is faster and requires less memory than explicit routines.
- The algorithm has been successfully implemented on a shared-memory architecture with only a small increase in work memory.
- Convolution algorithms are available for complex non-centered data and centered Hermitian-symmetric data in 1D, 2D, and 3D.
- Ternary convolution algorithms are available for centered Hermitian-symmetric in 1D and 2D.

Future work

- Develop a distributed-memory implementation based on openMPI.

Future work

- Develop a distributed-memory implementation based on openMPI.
- Add additional routines, such as convolutions on real data, self-convolution, correlations, etc.

Resources

FFTW++:
http://fftwpp.sourceforge.net

Asymptote:
http://asymptote.sourceforge.net

Malcolm Roberts:
http://www.math.ualberta.ca/~mroberts